(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
5 April 2001 (05.04.2001)

PCT

(10) International Publication Number
WO 01/24093 A1

(51) International Patent Classification[7]: G06F 17/60

(21) International Application Number: PCT/US00/26883

(22) International Filing Date:
29 September 2000 (29.09.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/157,350 30 September 1999 (30.09.1999) US
09/669,594 26 September 2000 (26.09.2000) US

(71) Applicant: INSTANTIS, INC. [US/US]; 913 Hermosa Court, Sunnyvale, CA 94085 (US).

(72) Inventor: RAJE, Prasad; 34336 Dunhill Drive, Fremont, CA 94555 (US).

(74) Agents: MALLIE, Michael, J. et al.; Blakely, Sokoloff, Taylor & Zafman LLP, 12400 Wilshire Boulevard, 7th Floor, Los Angeles, CA 90025 (US).

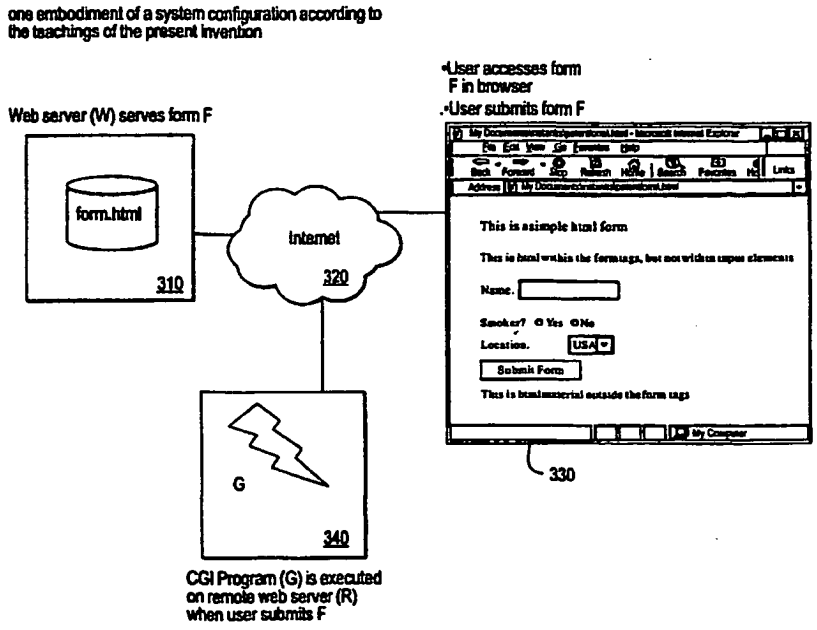
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*
— *Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: METHOD, APPARATUS, AND SYSTEM FOR ENABLING CREATION AND MAINTENANCE OF REMOTE CGI SCRIPTS ON THE INTERNET



(57) Abstract: According to one aspect of the present invention, a method is provided in which a form is received from an author. A parsing function is performed to extract attribute information with respect to various form elements contained in the form (see Figure 3, items 310, 320, 330, 340). A user interface is presented to the author to allow the author to configure a set of actions that are to be performed in processing the submissions of the form.

WO 01/24093 A1

METHOD, APPARATUS, AND SYSTEM FOR ENABLING CREATION AND MAINTENANCE OF REMOTE CGI SCRIPTS ON THE INTERNET

RELATED APPLICATIONS

This application claims the benefit of U.S. Provisional Application No. 60/157,350, filed September 30, 1999.

FIELD OF THE INVENTION

The present application relates generally to the field of information processing. More specifically, the present invention relates to a method, apparatus, and system for enabling the creation and/or maintenance of CGI scripts on the Internet.

BACKGROUND OF THE INVENTION

Hyper Text Markup Language (HTML) is a popular authoring language used to create documents on the World Wide Web (WWW). HTML forms are in widespread use on the Internet in order to receive input from users that visit a web site. HTML forms elements are a subset of the HTML specification (see for example HTTP://www.w3.org/MarkUp/). The complete specification for HTML forms is not recited here, but is incorporated by reference.

HTML forms can contain various input elements including the following:

- text fields (represented in HTML as <input type=text>)
- password fields (<input type=password>)
- radio buttons (represented as <input type=radio>)
- checkboxes (<input type=checkbox>)
- buttons like submit, reset and button (<input type=submit>, <input type=reset>, <input type=button>)
- select lists (<select><option></select>)
- text areas (<input type=textarea>)
- hidden fields (<input type=hidden>)
- file field (<input type=file>)

Figure 1 shows a simple example HTML file containing a simple form with some of the elements shown above. Figure 2 shows the same HTML form rendered in a modern browser. The input elements may have many different attributes. Usually, each of the input elements has a name that serves as a variable identifier. Collectively, the specific set of form fields in a particular form can be referred to as the "input fields" of that form. The set of names of the input fields of a particular form can be referred to as the "input field names" of that form. In the example shown in Figure 1, the input field names are: firstname, smoker, location, id, submit.

Users provide input into these form elements on a web page. A user's input in some field in the browser effectively becomes the value associated with the name of that field when the form is submitted to a web server for processing. The specific user inputs into a specific form are referred to as the "input field values" or "submitted field values" or the "submitted form values".

When an HTML form is submitted (e.g., by the user clicking on a <input type=submit> element or by causing the same effect as the pressing of a submit button using scripting language like Javascript or others), a Uniform Resource Locator (URL) on a HTTP server is accessed. The input field values in the HTML form are passed to the server using either a "GET" or "POST" method of passing these variables.
The HTTP specification (HTTP://www.w3.org/Protocols/rfc2616/rfc2616.txt) and the Common Gateway Interface (CGI) specification (HTTP://web.golux.com/coar/CGI/) describe the manner in which the variables are delivered to the web server and then to the program that is executed as a result. In the example in Figure 1, the method is POST and the URL that is accessed is HTTP://www.remoteserver.com/form.cgi.

The program on the server that receives the user input and performs some processing of the input and generation of a response is called a "CGI program". The term CGI program used here does not imply any particular language (which could be Java, C, shell scripts, Perl, Python, ASP, Tcl/Tk etc) or any particular server hardware or HTTP server software.

The CGI program can be located at an arbitrary location referenced by an URL as specified in the <form> tag of the HTML page. The tag contains an ACTION attribute that specifies the URL of the CGI program. Typically, the CGI program is on the same web server as the HTML form. But this is not necessary and the CGI program can reside

at an arbitrary URL. A CGI program that resides on a different web server from the HTML form is referred to as a "remote" CGI program. A CGI program that resides on the same web server as the HTML form is referred to as a "local" CGI program. The CGI program or CGI script can be a fully general program that is executed when a form submission is received. In order to write any arbitrary CGI program for any arbitrary form, it would require a human that is capable of writing the program and it would require this human to have knowledge of the form field elements and knowledge of what functionality the program is intended to have. One of the aspects of this invention is to eliminate the need for a human to write the CGI program. Thus, the CGI program is to be generated programmatically. In order to make this problem tractable, the CGI program cannot be expected to perform any arbitrary function. Instead, there is a super set of actions that this CGI program might perform. This superset can include performing some computation with the input field values to generate some dependent values, generation of a web page in response to the user, generation of various email messages which can contain some or all of the input field values or other content, storage of the submitted data in a database or other persistent store, triggering of change of state of the contents of a database or persistent store etc. These actions are described in more detail later. For any given form, the CGI program might perform some subset of the superset of actions. Thus, a specification is required that is the set of actions that the CGI program is expected to perform for a given form. These actions may or may not depend on the field elements of a given form. These actions may or may not depend on the field values entered by a given user in a given form.

CGI programs may be classified into the following types on the basis of the dependence of the actions performed by the CGI program on the specifics of the form field elements or the form field values.

- Type A: This is a CGI program that does not require any a priori information of any of the specific input fields in the form. It performs the same function, regardless of the specific input fields in the form. By extension, it performs the same function regardless of the specific submitted field values in the form. An example of this type of CGI program would be a CGI program that simply takes all the input field names and submitted field values and saves it into a file and provides a fixed response to the submitter. In the example shown in Figure 1, if the CGI program was type A, it might

simply respond to the user with an HTML page containing the string "Thank you". The program would respond in exactly the same way if the form was modified to include a few extra input fields, or if a completely different form had its ACTION URL pointing to this Type A CGI program.

- Type B: This is a CGI program that does not require any a priori information of most of the specific input fields in the form, but does rely on the existence of certain agreed upon input field names to customize certain aspects of the behavior of the CGI program. The "agreement" is between the author of the form HTML who determines the fields and their names in the form and the author of the CGI program that expects to use certain field names to control its behavior. In the example in Figure 1, the CGI program "form.cgi" may expect to see an input field name such as "emailto" in the form and uses the submitted field value associated with the "emailto" field to determine the email address to which to email all the submitted field values. Apart from these "agreed upon" field names, this type B CGI program makes no assumptions and has no knowledge of any of the other input fields or the input field names or the input field values. If the form html was modified to have a few new fields, but the emailto field was retained, then the type B CGI program would perform execution in exactly the same way for this new form. That is, it would send the email to the value submitted in the emailto field. Thus, a priori knowledge of all fields in the form is not required for the CGI program to perform its function. These arguments are not limited to emailing of form contents, the same arguments apply to other functions performed by CGI programs. For example displaying an html page in response to a submission etc.
- Type C: This is a CGI program that does require a priori information about the specific input field names of the form and whose behavior depends on a substantial number of these input fields. However, there is no functional dependence on the submitted field values of any of the fields. In the example in Figure 1, "form.cgi" would be a type C CGI program if it expected to store the user's input in a database table whose columns matched the fields in the form, namely firstname, smoker, location and id. The same function is performed regardless of the actual values of the submitted form fields. However, if the form was modified to have an extra field such as "Age", then the same "form.cgi" would not be able to store the user's input for

"Age" in the database table because there was no column defined for "Age" in the database table. Thus, "form.cgi" requires a priori information about the fields in the HTML form.

- Type D: This is a CGI program that does require a priori information about the specific input field names of the form (like a Type C program described above) and also whose functional behavior depends on the specific submitted field values input by a user during a particular submission. The CGI program needs to have knowledge of the specific field name that asks for the user's input and the user's input further determines a functional behavior of the program. In the example in Figure 1, "form.cgi" would be a type D CGI program if it examined the value of the user's response to the radio element "smoker" and if the input was "yes" - it responded with "You are wise to not smoke" and if the input was "no" - it responded with "The Surgeon General has determined that smoking may lead to cancer ".

One of the aspects of this invention is to enable the programmatic creation of CGI programs. A further aspect is to do this with minimal interdependence between the authoring of HTML form and the creation of the CGI program. The HTML form is not constrained in any way with respect to the input fields (their number of type etc) in it. The author of the HTML form does not directly create the CGI program.
At most, he is expected to provide some input to enable the programmatic creation of this CGI program. This input could contain specification of what specific subset of actions should be performed by the CGI program for the specific form instance. The input may or may not contain the html form itself since these actions may or may not depend on the input field elements of the form and may or may not depend on the input field values. Depending on the type of CGI program (A, B, C or D) the following situations may arise:

By definition, a type A CGI program performs a fixed action or set of actions with no dependence on the form field elements. Thus there is no interdependence between the authoring of the form and its associated CGI program. No specification of what action is to be performed by the CGI program needs to be provided since this action is fixed a priori. No knowledge of the specific input fields in the form is required by the CGI program since the action performed is independent of the input fields

In the case of Type A CGI programs, the solution may be simple. All forms that would like to avail of the functionality provided by such a CGI program can simply point to its URL in the ACTION tag of the <form> element. Of course, if some other type A CGI program is desired which performs some other (fixed) function then this can also be achieved with no dependence on the form field elements or field values.

In the case of type B CGI programs, the solution may also be simple. The functionality of the type B CGI program is also fixed for all forms. It is the responsibility of the HTML form author to learn about how the CGI program can be customized and insert appropriate directives in the HTML form with input tags of type <hidden> or other input tags. For example, suppose a CGI program performs the function of simply sending by email the user's input provided in the form. The destination address for the email could be specified by the CGI script author as a field in the HTML form with name=emailto. Thus, in the example shown in Figure 1, the HTML author could add in a field in the HTML form as follows:

<input type=hidden name=emailto value=author@company.com>

When the form is submitted by the user, the CGI program expects a name "emailto" and examines its value to determine the destination for the email. Apart from this agreed upon field, the behavior of the script would be substantially independent of the rest of the form. However, this approach cannot be the full solution because the HTML author should not be expected to learn about the CGI program's specification and to insert these specific tags in the HTML form. Also, not all CGI programs can be constrained to be type A or B.

In the case of type C or type D CGI programs, there has to be some way for the CGI script to learn about the specifics of the input field names and other relevant information in the HTML form. One method may be to have an integrated system where the form is authored and the functions of the CGI program are customized at the same time.

In this scenario, the specification of what actions the CGI program should provide (e.g., where to email the form, or which text file to put it into or how to validate the form fields, etc.) is configured during the authoring of the form. The information on the specific actions may be made available to the CGI program through a configuration file that is available to the CGI program. Another method may be to embed information

as comments within the HTML form and make the HTML form file directly available to the CGI program. Thus the CGI program is able to consult the configuration information via a file. When a submission of the form comes in, the CGI program knows what specific functionality is desired from this form.

Another method employed to customize the handling of the form might be to require the author of the form to create the form within a web browser that is interacting with a program that will directly generate the configuration information required for the CGI program.

However, the above methods impose a constraint on the author. It is preferable to not constrain the author to creating the HTML form in any particular front end authoring tool or to visit a web site to create the HTML form. It is preferable to provide the flexibility to the author of the HTML form to create the HTML in any suitable environment that is capable of generating the appropriate HTML for the form, even with the CGI program of type C or type D.

SUMMARY OF THE INVENTION

According to one aspect of the present invention, a method is provided in which a form is received from an author. A parsing function is performed to extract attribute information with respect to various form elements contained in the form. A user interface is presented to the author to allow the author to configure a set of actions that are to be performed in processing the submissions of the form.

BRIEF DESCRIPTION OF THE DRAWINGS

The features and advantages of the present invention will be more fully understood by reference to the accompanying drawings, in which:

Figure 1 illustrates an example of a simple HTML form in HTML code format;

Figure 2 shows an example of a simple HTML form rendered in a browser;

Figure 3 illustrates a block diagram of one embodiment of a system configuration according to the teachings of the present invention;

Figure 4 shows an example of one embodiment of a user interface for configuring form functionality;

Figure 5 shows a block diagram of one embodiment of a system for remote configuration of forms and generation of corresponding CGI programs; and

Figure 6 is a flow diagram of one embodiment of a method according to the teachings of the present invention.

DETAILED DESCRIPTION

In the following detailed description, for purposes of explanation and illustration, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be appreciated by one skilled in the art that the present invention may be understood and practiced without these specific details.

In the discussion below, the teachings of the present invention are utilized to implement a method, apparatus, and system for creating and maintaining remote CGI scripts for processing forms. In one embodiment, a form is received from an author. A parsing function is performed to extract attribute information with respect to various form elements contained in the form. The author of the form is allowed to configure a set of actions to be performed in processing instances of the form submitted by one or more users. In one embodiment, a user interface is provided to the author to allow the author to specify the set of functions to be performed in processing instances of the form. The user interface is constructed based upon the attribute information extracted from the form. In one embodiment, the user interface includes a set of questions constructed based upon the extracted information from the form. The set of questions included in the user interface is used to obtain information from the author with respect to the set of actions desired by the author in processing the form. In one embodiment, the configuration information provided by the author via the user interface is used to configure a specific program including one or more program components to perform the one or more functions specified by the author in processing the form. This specific program is designated as the program to process instances of the form submitted by the users. In one embodiment, the URL of this specific program is then provided to the form author so that it can be inserted into the ACTION attribute of the <form> tag in the author's original form html. Alternatively, the author can be provided with a "modified" form html with the ACTION tag already inserted so that the author can simply replace the original form html with the modified form html. This enables a submission to the form to delivered to the specific program that has been configured by the user to process submissions to this form. In another embodiment, a set of directives corresponding to

the set of functions desired by the author is generated. This set of directives is to be used by a generic form processing program designated to process multiple forms including the first form which is authored by the first author. The generic form processing program is configured to process multiple forms based upon multiple set of directives where each set of directives corresponds to a particular form authored by a particular author. In one embodiment, a URL is provided to the author that can be inserted into the ACTION attribute of the <form>. This URL points to the generic form processing program, but in addition has an identifier that uniquely identifies this form and the set of directives that have been configured by the author. This enables a submission to the form to be delivered to the generic form processing program, which can then examine the unique identifier and execute the specific set of functions configured by the author.

In one embodiment, a configuration data structure is generated based upon the configuration information provided by the first author using the first user interface. In one embodiment, the consistency between the configuration data structure and the first form is maintained. In one embodiment, the configuration information contained in the configuration data structure is modified accordingly in response to changes in the specification of the first form. In one embodiment, it is determined whether the first form has been changed since the configuration data structure corresponding to the first form was created. If the first form has been changed since the configuration data structure was created, the configuration data structured is updated to reflect the changes that have been made to the first form. In one embodiment, the configuration data structure is modified based upon the changes to the first form that are provided by the author. In another embodiment, a timestamp associated with the first form is maintained to determine whether the form has been changed. The timestamp associated with the first form is periodically checked to determine whether the first form has changed. If so, indicate that the first form and the configuration data structure are out of sync. In another embodiment, it is determined whether the form has been modified with respect to the form elements contained in the form. If so, indicate that the form and the configuration structure are out of sync. In yet another embodiment, maintaining consistency between the form and the configuration structure is performed at the time when an instance of the form submitted for processing. In this scenario, the current instance of the form elements in the form are extracted and compared with previous

instance of the respective form elements stored in the configuration structure. If the current instance and the previous instance of the form elements are different, indicate that the form and the configuration structure are no longer consistent.

Figure 3 illustrates a system configuration according to the teachings of the present invention. As shown in Figure 3, in this configuration, it is assumed that a form F is authored by an author A using any HTML authoring environment on some computer C. In one embodiment, there should be no constraints imposed on the number, nature, names of the fields or surrounding HTML text and tags in F. It is also assumed that the form F is to be served on the web by a web server W (W may be the same as C). In one embodiment, F may be referenced by its corresponding URL (e.g., HTTP://W/F). It is further assumed that the CGI program G for this form is "remote", that is, it is on a distinct web server R. As a degenerate case, it is possible for R to be the same as W, but it is not a necessity. In one embodiment, the location of the CGI program G may be referenced by its corresponding address in URL format. Accordingly, the target URL for the ACTION attribute in the <form> tag of the form F may be specified as HTTP://R/G. Thus a user U that accesses F, fills it out, and submits it, will cause the URL HTTP://R/G to be accessed and cause the CGI program G to be executed. It is assumed that the author A has no ability or desire to write the specific program code inside CGI program G. Therefore, it is assumed that the CGI program G needs to be generated programmatically, without human action, on a computer distinct from C. It is also assumed that during the authoring of form F, there is no information about F made available to the program that generates the CGI program G. Further, G is not constrained to be necessarily a Type A or a Type B CGI program. That is, G does not necessarily have a fixed behavior regardless of the details of form F (i.e., it is not necessarily a type A) and there are no agreed upon fields to be placed in the form F (i.e., it is not type B). G may be a Type C or Type D CGI program, one whose functionality depends on the field names in F and possibly on the submitted field values. It is further assumed that there is no restriction on author A with respect to changing any part of form F at any time.

In the discussion below, the teachings of the present invention are utilized to provide a solution to the problem described above.

Figure 4 illustrates a block diagram of one embodiment of a system configuration to allow an author A of a form F to configure a remote CGI program G that will be used

to process instances of the form F. As shown in Figure 4, the author A may use a web browser to open a web page on a computer N. N is distinct from C, but could be the same as R. The author A then provides to N the URL of F (e.g., HTTP://W/F). Alternatively the author A might upload form F to N directly.

In one embodiment, a parser program P on N fetches F and performs the function of parsing F. The parsing function includes extracting information from the input field tags in F. This information may include a list of the fields, the field types, the field names, values of other attributes of each field etc. In one embodiment, the parsing function also includes creating a "signature" of all the text within the tags of the HTML form that correspond to user inputtable fields. In one embodiment, this signature of form F is stored along with other information about F including file modification date, if available.

Continuing with the present discussion, in one embodiment, a configuration program U takes the output of parser P, and dependent on the fields in F, presents to the author A, a user interface through the web browser. The user interface may be a Java applet or an HTML page with form fields for the purposes of customization. An example of one embodiment of the user interface is illustrated in Figure 5. In one embodiment, the user interface as shown in Figure 5 is used by the author A to configure a set of functionalities or functions that are desired by the author A to be included in the CGI program G. In this example, it is assumed that the CGI program G is the processing program for form F. An illustrative set of various functionalities that may be configured according to the teachings of the present invention is described in more details below. The knowledge of the form as represented in the output of parser P is used by configuration program U to ask only relevant questions that are presented to A in the user interface. For example, if G is required to do validation of submitted field values, then U knows to ask for string validation for fields in form F that are of type text <input type=text>. Likewise, if there is a field with name matching "email" then U can make intelligent guesses and ask A if the field should be checked to contain an email address, etc. In addition, U also presents to the author A, various questions relating to various functions that are to be performed in G, including, but not limited to, emailing the submitted form fields, saving the submitted form fields in text files/directories or databases, generating cookies, performing computations on the input fields of F, taking

special actions based on submitted field values etc. As such, the author A is provided with the capability to configure the various functionalities for the CGI program G with respect to the processing of form F from a browser based environment. In one embodiment, the configuration program U may include several separate programs, subprograms, or program components, each of which is designed to configure one or more functions that are to be included in the CGI program G.

In one embodiment, the output O generated by the configuration program U may be a CGI program that may include various code modules (also referred to herein as program components). In one embodiment, each code module may be configured to correspond to a functionality area or function to be performed in processing form F (e.g., response to user, emailing the form etc.). The module may be further customized based on the specific configurations performed by the author A. In this scenario, there is a separate program G for each separate form F and set of configurations for F. Alternatively, in another embodiment, the output of the configuration program U may be a set of configuration directives for a general purpose CGI program. In this embodiment, G can be configured to be the same processing program for all forms F. However, the path of execution through G is different for each form F and its configuration settings. Thus, the course of execution through G for each form F and its configuration is controlled by the corresponding set of configuration directives generated by the configuration program U. For example, if an author A1 configuring form F1 did not want to perform any validation of the form inputs, the validation module would not be executed because of the appropriate switch statement (or other similar mechanism). Likewise, another author A2 configuring form F2 who did not want any emailing of the form would result in the configuration directive containing a setting that would prevent the execution of any code in G relating to emailing of the form. Another author A3 configuring F3, with a finer degree of configuration within the form emailing feature relating to the format of the email would end up invoking appropriate sections of code within G for controlling the format of the email, etc. When the form is submitted and G is executed, additional information needs to be provided to G such that it can access the specific configuration for the form being submitted. This might be done with an extension to the URL of G. For example the ACTION attribute in the <form> tag would be http://R/G/F1 when form F1 is submitted or http://R/G/F2 when form F2 is submitted.

This will cause G to be executed, but per the CGI spec, G will have additional information indicating that the extension F1 was present or the extension F2 was present, allowing it to look up the appropriate configuration directives for F1 or F2. This could also be achieved by other means like providing a <input type=hidden> element in the form. For example <input type=hidden name=formid value=F1> in form F1, and <input type=hidden name=formid value=F2> in form F2. Thus when F1 is submitted, it will come with a formid=F1 and G will be able to access the configuration directives for F1 and execute the actions that the author configured for F1.

As such, these configuration directives are specific to each form F. The advantage of this approach is that there is only one CGI program G to maintain and develop. Execution efficiencies are also possible because only one copy of the program needs to be in memory even if there is a large number of simultaneous submissions of many different forms.

In one embodiment, the output O of U represents the set of functionalities or functions desired by author A in the corresponding CGI program G that is designated for the processing of form F's submissions by the users. In one embodiment, the output of U may be represented as a configuration file or configuration data structure. In one embodiment, it is possible for the author A to be taught about the syntax of the configuration file that is the output of U. Then the author A might provide the configuration information directly, without the help of a user interface. The configuration information may be provided by the author A in the form of a simple text file that contains directives that are used for configuring the actions or functions of G. In one embodiment, the configuration program U might also provide for the configuration of other items beyond the configuration of G. For example, it might provide for the configuration of how the cumulated form submissions are to be managed which is not related to the actions that take place in G during the actual submission of the form. Once the author has configured the desired functionalities of the CGI program G, the original form of the author needs to be set up such that submissions to the form do indeed invoke G. This can be done by simply ensuring that the ACTION attribute of the <form> tag in the form F points to the appropriate URL. This is http://R/G with possibly extra identifiers in the URL as described above. The author may choose to simply edit the form to insert this URL or the author may be provided with a "modified" form which

contains the appropriate URL already inserted into the ACTION attribute of the form or other appropriate insertion of an identifier into the form html. This would enable the author to simply replace the original form F, with a substantially identical form that is set up to invoke the remote CGI program G upon submission.

The present invention described herein also provides a solution to the problem of change management with respect to the form F described above. The CGI program G that is created after the configuration might be a Type A/B or a Type C/D program. If it is a type C/D program, then any changes made by author A to the form F could have a potential impact on the correct functioning of G.

The problem associated with the change management of form F is illustrated with the following example. In the example form in Figure 1, suppose the CGI program is expected to respond with a set of instructions which depend on the user's location as selected in the "Location" select input. Suppose the form author changes the form such that an additional location is added, e.g., to change the HTML in the select input to Location:

<select name=location>
<option value=US>USA
<option value=Asia>Asia
<option value=ROW>Other
</select> <p>

Then the corresponding CGI program designated to process form F will not be able to correctly provide the needed response for the Asia location. Thus a change in the form can lead to incorrect operation of the CGI script if it is kept unchanged.

According to the teachings of the present invention, several levels of solution may be implemented to solve this problem. These solutions include:

1. Asking the author A of F to communicate to the configurator (i.e., the configuration program U) that F has changed and going through some part of the configuration process again to make the appropriate changes to the CGI G. One disadvantage with this approach is that it may be error prone if the author A forgets to do this.

2. Keeping track of the time stamp of the form F. When the CGI program G is first configured for F, the timestamp and file size of F is stored. In one embodiment, a

periodic process called a "consistency checker" CCP1 running on the remote web server R or on a related computer, could periodically check the timestamp of F to see if it has changed compared to the stored value. One method of checking the timestamp of F is to use the HTTP protocol to check the "Last-Modified" date of the form html. In one embodiment, if a change is found, then this would set a "out of sync flag" (OF1) that would indicate that F is out of sync with G. When the OF1 is set, unpredictable results can occur if a submission is received by G. The execution could proceed properly or there might be too many changes in F for G to handle. In general, this would be difficult to ascertain a priori. Thus one action for CCP1 to take could be to send an email or other notification to the form author indicating that the form and its configuration were out of sync. One action that the author could take then would be to redo the configuration with the changed form.

A limitation with the approach described above in solution #2 is that changes to the form F would trigger the out of sync flag even when there is no need for a change in G. For instance, in the example HTML form of Figure 1, if the author A simply edited a spelling mistake in the HTML outside the <form> tags, or if the author added in some explanatory text like "Please select a Location" instead of just the word "Location:", then there would be a change to the timestamp and/or file size of F. This would trigger the "out of sync flag". However, the functional behavior of G is really not changed because of this change in the form.

3. An improvement over solution #2 would be to set the "out of sync flag" with a finer granularity of change. One method would be for the consistency checker actually fetching F using its URL http://W/F. Then the internals of the html in F can be examined for changes. In particular, the change in F that might be relevant to G can be considered to be limited to changes within the input field elements of the form. These changes include:

- Addition of a new field element. In the example of Figure 1, this might be the inclusion of a new text field such as:
  Last Name: <input type=text name=lastname size=10><p>
- Removal of an existing field element. In the example of Figure 1, this might be the removal of the text input for firstname.

- Change to an existing field element. This covers any change to an existing input element, in its name, its possible values for those elements which have a finite set of possible values (elements like radios, checkboxes, selects etc.), its visual representation like size or other attributes of the input element. In the example of Figure 1, inclusion of an additional response for the Smoker question is a change to an existing field element.

Smoker?: <input type=radio name=smoker value=yes> Yes   <input type=radio name=smoker value=no> No   <input type=radio name=smoker value=former> Former smoker<p>

Thus the consistency checker CCP1 would be enhanced to check the form F not just for timestamp or file size changes, but finer granularity changes that could potentially affect the operation of G. This finer granularity checker is referred to here as CCP2. When CCP2 detects a change in F, it sets a out of sync flag OF2. When OF2 is set, G could continue to service submissions to the changed F under certain circumstances. For example, with any new field element, G could perform certain actions like including that element in an email that is sent or create an extension to the database that is to store the elements. It would however not be able to perform certain other actions like doing validation checks on this new field element. With removal of a field element, G would not be able to include that element in an email or insert it into the database, but could otherwise continue to perform its actions as initially configured. With change to an existing field element, G could also continue to perform certain actions like emailing the values or inserting them into the database. There might be certain actions that fail however – for instance if there was a validation check that was set up to ensure that a certain element had only a subset of a fixed set of values, then this validation check would fail. If a change to F were to result in an incompatible change to G, CCP2 could take the action of sending an email or other notification to the user. The user could then redo the configuration for the changed F.

4. The solution #3 described above achieves finer granularity checking but requires a continuous, periodic program CCP1 or CCP2 to check the form F. This could be expensive in terms of computational power and bandwidth, especially if a large number of forms F need to be checked. Also, if the form F is changed and submitted to G in the interval between periodic runs of the checker CCP1 or CCP2, then a problem could

result from G being out of sync with F for an interim time period. Conversely, if F does not change very often, then the consistency checker CCP1 or CCP2 would be run without much benefit most of the time.

Another solution to the consistency problem is to perform the consistency checking dynamically - that is when the form is actually submitted to G for processing. This approach works as follows. In one embodiment, when the form is submitted to G, the input fields in the form are made available to G as per the HTTP and CGI Specifications. G then invokes a modified consistency checker CCP3. In one embodiment, CCP3 is similar to CCP2 in that it is concerned with a finer granularity of changes in the form F (i.e., changes in the HTML within the input fields). In one embodiment, instead of taking a complete HTML file as input, CCP3 takes as input the actual fields that are received in the user submission, per the HTTP and CGI specifications. In this manner, CCP3 is not dependent on any changes in the form F that lie outside of the input fields since these changes will not manifest themselves in the data that is submitted to G by the browser per the HTTP and CGI specifications.

During the parsing process (described above for parser P), the list of input fields in the form F and the set of allowed values for these fields are stored. CCP3 then examines the input field names and the values received when form F is submitted and compares them to the list of input field names and values that have been previously stored. If a change is detected in the input field names, the CCP3 sets the "out of sync flag" OF3. The changes that CCP3 might detect include:

- Addition of a new field element. In the example of Figure 1, the inclusion of a new text field like

  Last Name: <input type=text name=lastname size=10><p>

  will result in the G receiving an additional field "lastname" along with the other fields received from the user's browser. This is detected by CCP3 as not being present in its prior list of known field names for the form.
- Removal of an existing field element. In the example of Figure 1, this might be the removal of the text input for firstname. In this scenario, the field firstname will not appear as one of the fields received from the user's browser. CCP3 can flag the fact that an element is absent from the form. It is possible that this absence from the form could cause an error in G, for example if G depends on the firstname field to check

for duplicate submissions. However, it is also possible that, the functionality of G can proceed without causing any error, for example if G is simply storing the data in a database table, the table might be allowed to have empty fields in certain columns.

- Change to an existing field element. This covers any change to an existing input element, in its name, its possible values, its visual representation like size or any other attributes of the input element. In the example of Figure 1, inclusion of an additional response for the Smoker question is a change to an existing field element. This would be represented in the form HTML as

Smoker?: <input type=radio name=smoker value=yes> Yes   <input type=radio name=smoker value=no> No   <input type=radio name=smoker value=former> Former smoker<p>

In one embodiment, if the user does not choose the new value=former when submitting the form in the browser, then the checker module CCP3 will not be able to detect any change to the form. This is acceptable since there is no change to any input to the CGI program G. If the user does choose the new value=former choice in the radio button during submission, then CCP3 will see, for the name smoker value, a new value "former" that is not in its prior stored list of values associated with "smoker". This will cause CCP3 to raise the "out of sync" OF3 flag. It is possible that the execution of G can still proceed without error even with the new value of field smoker. If is also possible that there would be an error in some part of the execution of G. In this case, CCP3 might notify the author of the form by email or other means. The author might then take the steps of going through the configuration of the actions again to ensure that the out of sync condition is removed.

The following contains a description of various types of functionality or features that the CGI program G can be configured to perform upon receipt of a submission of a form from a user. These various functionalities may be called the "submit-time" functionalities.

As described above, specification and details of each functionality can be configured by the author through a web based interface as shown in Figure 5. Not all features will be relevant to each form. Any author may choose to use any of these features as desired.

The following list is an illustrative list of the various types of functionalities that CGI program G can be configured to provide and the types of configuration parameters that the configuration program U needs to obtain from the author of the form.

- Form field validation: when a user submits a form, each field in the form can be validated against a rich set of tests. For example, a phone number field may be checked to ensure only digit or allowed separator characters, have a particular length etc. Another check might be to ensure that text fields do not have "random input" in them like "asdf" or "xxx" etc.
- Derived quantity generation: based on the user input, a derived quantity may be computed or constructed. For example, if two items are selected, their total quantity can be computed.
- License generation: a random license key can be generated in response to a user form.
- Cookie generation and tracking: a cookie can be created and set for the user. Any existing cookies already set can be included in the forms handling by looking up the cookie in a stored database or file.
- Form emailing: each form as submitted can be emailed to the customer with user filled contents in a variety of formats. The form might be emailed to fixed destination(s) or to selectable destinations, based on user input. Example, if a user expresses interest in product A (instead of product B) and the sales person handling product A is different from that handling product B, the email destination for the form would be sales person A and not B.
- Email to user: in response to a form submission, an email message can be sent to the user. For example, if a user signs up to receive a newsletter, the response can provide instructions on how to unsubscribe from it. This email can further be dependent on some input in the form.
- Form response: in response to a form submission, a next page is displayed. This can be customized to the customer form and to the users input. For example, the response page can contain some reference to the user's name or to something the user ordered by providing an input in the form.
- Form threading: many forms are multi-part forms and state needs to be preserved between forms. Threading can be set up to ensure certain form state is passed

between forms. For example, some resume submission forms might require a prequalification form to ensure the user has some basic qualifications. The subsequent application form could carry forward this information from the previous form so it does not have to be rewritten by the user.

- Form logging: each form can be accumulated in a log along with other submissions of the same form by other users. Multiple log formats are supported, including single files per submission, cumulated files with each submission appended, file and directory structures created based on calendar and time, etc.
- Special events: when a form is submitted that meets certain special criteria, or certain criteria are met for cumulated submissions, a special action or alert can be set up. For example, if the user has a title of CEO and the company size is >500M, then the form can be set up to be emailed directly to the VP of Sales.

The above list of functions is meant to be illustrative and other functions that CGI programs are expected to perform can similarly be included.

In one embodiment, there is a customer log in script (e.g., "login.CGI") that is used to admit a customer to the administration interface for his forms. In one embodiment, the login.CGI may output a page that contains the dynamically generated list of this user's active forms. From here, the user can select one of the forms or choose to activate a new form. In one embodiment, this runs a form configurator program or form configurator script (for example, named "formconfigurator.CGI"). In one embodiment, the formconfigurator.CGI presents the full list of the customizable functionality for this form (validation, emailing, logging etc). From here, the customer can choose to customize each of these functionalities described herein.

In one embodiment, there is a form parser script (for example, named "parser.CGI") that parses the customer form and creates a representation of the input fields of the form. The representation could be some textual format or perhaps some language specific (e.g., Perl or Java) data structures that can be included by other scripts. In one embodiment, the parser.CGI also creates a database table for this form, with table columns being form input fields. The table might also contain spare columns for later use.

In one embodiment, the teachings of the present invention can be utilized to provide for customization by advanced users that do not require going through the interactive

browser interface. This might include exposing the structure and syntax of an intermediate configuration file that can be written to directly.

Form Field Validation

The form can be designed to ensure proper user input into the fields, including required fields, simple validation of field contents etc.

Each field in the form can be validated against a rich set of checks. In one embodiment, the standard input types in HTML forms are supported.

- Text fields: a text field can be forced to be:
  - Non empty, having a particular range of lengths
  - Beginning with, ending with or containing a fixed string. For advanced users, a full regular expression check can be done.
  - Containing a well formed item like: email address, telephone number, zip code, credit card number, name of US state, name of country, etc.
  - Not containing "keyboard noise" like aaaa or asdf or fggh etc.
- Password: a password field can be forced to have a particular length range, have a certain number of non-alphanumeric characters, not be equal or similar to another text field etc.
- Checkboxes: a checkbox is usually an optional item but it can be forced to be checked. More advanced functions might include requiring the checkbox if some other precedent condition is met.
- Radio buttons: a selection of one item of a group of radio buttons can be enforced.
- Select boxes: the user can be forced to select any item except the default item.
- File: The file can be forced to be non-null, have a certain extension, contain a string within the name, etc.
- Text area: the text area can be forced to have a certain string or regular expression, have a range of length, etc.
- Hidden fields: a form can have any number of hidden fields. Often hidden fields have special meaning to the forms functionality. As a security measure these fields can be forced to have fixed values or match regular expressions.

In one embodiment, each field that is validated can be named (the default is the name of the form field) and a custom error string for this field can be entered by the

customer (a default can be provided). This is the string that is used when an error message is returned to the user who is filling in the form.

In one embodiment, field validation is performed each time on all elements of the form, not on each element sequentially, so that error information for the entire form can be provided at one time.

In one embodiment, the error returned to a user when a form field validation fails can be of the following types:

- A fixed HTML page is returned, indicating that the form was not complete and suggesting that the user go back to the previous page. A standard default "Incomplete Form" page is provided, but this may not match the look of the original form on the customer site. This page may be best designed by the customer or author.
- A fixed HTML page but with specific information about the outcome of the validation checks inserted into a predetermined spot is returned.
- Some type of window pop up may be implemented which would keep the original form in the browser, but provide the validation results in a second, smaller, minimally embellished, browser window.

In one embodiment, the entire set of customization for validation of the form can be named and saved in a file that may be referred to as a "validation file". Thus a user might set up various different validation requirements for a single form. This allows a whole different set of validation settings to be applied by simply naming a different validation file.

If an email address is required in the form, this can be enforced. Further, the address can be checked heuristically to ensure it appears to be an email address.

In one embodiment, the validation interface presents each input field in the form in sequence and depending on the field type presents the set of options for validation for each field. In one embodiment, each field can be named, its error string can be customized, etc, through a web form interface.

The customer may also set up the "Form Incomplete" error handling preferences. In addition, the customer may also set up the "Confirmation page". This is the page that is presented if the form submission is considered validated and proceeds without errors. The customer may also set up an "Error" page for an error situation.

In one embodiment, if validation is to be customized, a script named for example "customizevalidation.CGI" is run to provide the author with a mechanism to customize field validation. In one embodiment, there is a script that creates the validation form (named for example "validationformmaker.CGI") for the customer form, based on the output of the parser.CGI program and its internal knowledge of the types of constraints possible for the different field types.

In one embodiment, customizevalidation.CGI first runs parser.CGI and then runs validationformmaker.CGI. In one embodiment, there is a script that may be referred to as "validationfilemaker.CGI" that accepts the customer's settings for validation and generates a persistent file. In one embodiment, this file contains validation directives specific to the customer form and validation requirements. Multiple different validation files can be maintained for a single form. In one embodiment, the validation file that is currently applied may be referred to as a "current validation file". The format of the validation directives could be a formatted text file, possibly XML, or it could be language specific code (Java, Perl etc), or a mixture thereof.

In one embodiment, the CGI script that is run when a user submits a form calls, among other things that are described below, a validator script (e.g., "validator.CGI") that validates the user input versus the directives in the current validation file.

Derived quantity generation

The user input in a form can be manipulated and combined to create "derived quantities". These derived quantities can be used as if they were user input for the rest of the form functionality. Below are illustrative examples of the types of derived quantities:

- Arithmetic quantities: any mathematical formula can be applied to a form input. For example, the total amount in an order form can be computed as the sum of the cost per item, which can be number of units times the rate per unit.
- String quantities: any number of user inputs can be put together using string functions to create larger strings. For example, Full name from First Name and Last Name fields.
- Array quantities: a user inputs (e.g., in a select field) can be an array subscript in a simple array or a key in an associative array. The resulting value in the array is the derived quantity.

- Random quantities: a random quantity can be generated. This can be used to provide for state continuity which is also described with respect to the form threading functionality.

In one embodiment, each derived quantity may be given a name such as a variable name. Some of the examples with respect to the derived quantity functionality are provided below.

- First Name and Last Name text boxes can be concatenated together into "Name".
- The sum total of charges for multiple items purchased can be computed, the total with tax/shipping and handling can be computed. This can then be displayed in the response to the user.
- A select box might return values of 0,1,2. These values could be indexed into an array that associates 0 with "Disagree", 1 with "Agree" and 2 with "Don't know". This functionality can also be used to select an element of a predetermined array at random. For example, if there is a fixed set of passwords, one of them can be handed out at random to a user.

License generation

Many situations require a license or a key to be generated in response to a form. This can be used as a password or code or unlock key for a download, etc. The description below is an illustrative list of the functionality variants:

- Certain constraints may be set before a license can be generated. These are generally the validation constraints discussed above.
- The license may be a random string, numeral or other quantity like a public/private key pair.
- The license may be selected (e.g., randomly/sequentially, etc) as one of a set of pre-made license keys.
- The license length can be specified
- The license may have some form of date encoded, eg., 052999.1132
- The license, once generated, may be stored as a cookie with the user
- The license may be stored along with the rest of the form data in a form log
- The license may be stored in a separate log
- The license may be emailed to the user who submitted the form
- The license may be displayed to the user in the response page

As an example of this functionality, a software company distributes software on the net which is downloadable from the site. However, this software company requires a license key to be enabled. In this instance, the license key is provided to the user by email after the necessary fields in the form have been obtained. In one embodiment, the list of keys is taken from a table of keys uploaded to the system by the software company.

Cookie generation

This functionality enables the customer to create cookies for each user that has submitted a form. Prior cookie values that have been set can be picked up from the user. The functionality variants are as follows:

- The cookie string can be customized including its length, content, fixed parts, variable parts, random parts, etc.
- The cookie parameters can be customized including the URL, the expiration, etc.
- The cookie can be stored in the form logs

As an example, an online store that wants to ensure that repeat customers are offered some special items can check the cookie value set for each user submission and provide special responses to that user.

Form emailing

When a (optionally validated) form is submitted, the form contents can be emailed to an address or a list of addresses specified by the customer.

The list below is an illustrative list of the ways in which the email contents and formatting can be controlled.

- The subject line of the email can be customized
  - The subject might have a fixed text string
  - It might have a variable string composed from the name of the form, the time stamp, the submitter IP address/hostname, etc.
- The content of the body might be empty. If non empty, it might be a MIME attachment or it might be in line plain text. The MIME attachment or text might be formatted as HTML or plain text or other.
- The format of the content can be customized as follows:
  - Each field in the form can be optionally included or excluded from the email. The default would be to include every field.

- Additional information not directly entered by the user can be included in the email. This might be time stamp, IP address/hostname, browser version info etc.
- There could be a fixed header string and footer string above and below the list of name,value pairs submitted by the user.
- The sequence of name, value pairs can be adjusted to be in "form order" or "random order" or "alphabetically sorted by field name".
- Each name, value pair appearance can be customized to include/exclude the field name, to separate the name and value by a string, to stylize the field name by making it bold, etc.

- The email destination can be customized. There can be a fixed destination(s) or variable destinations. The destination could depend on the user input in some form field (eg., if a user is submitting a tech support form asking about product Z, the form is emailed to tech support representative Joe who is an expert in Z).
- The email delivery can be controlled to be instantaneous or delayed to a later time.

This functionality is expected to be a very widespread functionality of the hosted forms. For example, sales leads from customer inquiries can be sent to a sales email address in the company. Tech support inquiries from online forms can be sent to the tech support alias. A sales inquiry can be farmed out to separate email addresses, depending on the territory derived from the user's address. A tech support form can be sent to a specific person with the requisite skill set based on the product selection.

Email to user

After a form is received from a user, the user can be sent an email message. Below is an illustrative list of the variants of this functionality:

- The email format can be inline plain text or 1 or more MIME attachments. For example, a public relations firm might have an entire press kit consisting of many files sent in response to an inquiry.
- The emailed file(s) can be the same for all users, or different files may be selected depending on user input in the form. For example, if a user asks a telephone company for a calling plan information, he will get the relevant file depending on the state in the address.

- The email content could be fixed or customized to the user "mail-merge" style. For example, an email to a Mr. Smith who requested a bank statement for Jan 99 would start with "Dear Mr. Smith, Your request for a bank statement for Jan 99 is now being processed." If a Mr. Jones requested a statement for Feb 99, his email would be appropriately customized. The merging can be done with any of the user submitted variables in the form.

As an example, a user fills in an application for a job posting on the web site. A courtesy email indicating that the form is being forwarded to the right department manager can be sent to the user. As another example, a free home pages site receives an application from a new user. In this case, an email containing detailed instructions can be sent to the new user, along with a login name and password.

Form response

In one embodiment, the form response is an HTML page that is generated in response to a form submission. This is not the response to a user when the form fails a validation check as described above. This is generally a response to acknowledge or confirm the user's submission of the form (e.g., "Thanks for filling the form" response).

Below is an illustrative list of the variants of this functionality:

- The response may be a fixed HTML page regardless of user input. This is the simplest case.
- The response may be one of N static HTML pages that is selected based on some user input. For example, if a user submits a complaint form, then depending on the severity of the complaint the response HTML might be "You will be contacted to resolve the situation within 1 hour" or might be "You will be contacted in 3 days".
- The response may be "mail-merge" style HTML that is customized for the user submitting the form. For example, "Mr. A, Thank you for your application to the B Council" might be a customized response. The customization variables are given values based on user input in the form and derived values in the script.

Form threading

Many forms are designed to be used as one of a sequence. For example, a user might be asked some qualifying questions and then be presented with a follow on form. Some or all the input from the previous form can be retained in the second form as

hidden fields. The hidden fields can be inserted into the response page, just below a form tag.

For example, a user fills in his contact information and some skills information at a jobs site. Based on the skills, the user may be presented with a form to submit a resume or view some specifics of job descriptions. The resume form need not ask for contact information again if the input from the first form is already available to the second form. Below is an illustrative list of the variants of the functionality:

- Any user input or hidden fields from a prior form can be chosen to be threaded over to the response for this form.
- A derived quantity can be threaded over. For example, the sum total of an order can be carried into a second form that asks for payment information
- A random number derived quantity can be threaded over. For example, state can be maintained between multi part forms by carrying the random number first generated through the sequence of forms.
- A license key can be threaded over.

Form logging

One of the important functionalities provided is the comprehensive logging of the user submitted data. The purpose is to archive the forms contents for future reference, analysis, export etc.

Below is an illustrative list of the variants of the functionality:

- The form log format can be specified to be a file as follows:
  - The destination of each submission may be the same file. The log name is customizable. Within this file the submissions may be cumulated as follows:
    - A single submission is a row of comma separated values, with or without field names
    - A single submission may be in many lines, each submission is separated by a separator line. Each line is one field, with or without field names. Separators between records can be customized.
  - Each submission can go into a separate file. The directory and file name can be customized (for example to include a timestamp or alphabetical with the first name, etc). Within each file, the submission can be comma separated or multi line, as above.

- When the data is stored in a file, the data can be in many different formats - that is as plain text, as HTML, with various separators/embellishment of form fields and values etc.

- Database: each form can be defined in a database table (e.g., SQL database table) with the fields being table columns. Each submission can be a row in the table. The columns of the database table can be automatically created from the list of fields in the form. The types of the columns can be determined based on the types of form elements – example "string" type of column for <input type=text> etc. The relevant SQL statements to create the table would be issued when the form is being configured. The SQL statements that would be used to actually insert form submission data into the table could be prepared in template form, with only the values remaining to be inserted into the SQL statement when the form is submitted. .
- For each submission the items to be stored can be chosen. These are:
  - Some or all of the fields in the form, including hidden fields
  - Some or all the derived quantities, licenses, cookies, timestamps, etc
  - Some or all the extra associated information per form that come in the CGI environment variables (e.g., remote machine, browser information, etc.)
- Multiple logs can be specified for each form, with separate configuration settings for all of the above. For example, a mortgage broker has a signup sheet where users sign up to receive an email newsletter. The user information can be captured in a full database (e.g., SQL database, etc.). The user's email address is sent to a separate simple flat file containing email addresses. This file can then be used simply as an email mailing list file.

Database Functionality

The set of actions that take place in the forms handler when each form is submitted is referred to as the "submit time" functionality. The set of functions that is available to a customer with the database of submissions is referred to as the "database functionality". In one embodiment, the database functionality may include:

- Form logs management: this includes rotation, emailing, export, etc.
- Cumulative analysis: the user submissions in a form log can be analyzed interactively on demand. Pre-packaged analyses can be set up to run on a regular basis.

- Per submission access and update: individual submissions can be viewed or updated. Furthermore, certain pre-defined fields can be associated with each submission, where the fields are not in the form initially. These fields can be updated as desired.

Form logs management

In one embodiment, the log files or database tables that are used to cumulate form submissions may need to be managed by the customer.

- The logs can be configured to be rotated with a certain regimen. This could be based on a size limitation, a time/day/date driven regimen, etc. The rotation can also be done on demand.
- When a form is edited to add or remove fields, then there may exist an issue of how to deal with the previously gathered data. One option is to create a separate, distinct storage location (e.g., file, directory or database table such as SQL table) for the changed form. Another alternative is to continue to use the same destinations for the changed forms. The customer may need to resolve the ambiguities or differences in table widths, etc.
- The logs can be emailed to a specified address on a regular basis.
- The logs can be exported to other database or contact manager formats on demand. In one embodiment, supported formats could be Microsoft Access mdb files, Goldmine contact manager format, ACT format, etc.
- Another option is to have the forms database exposed to the customer through a "live" ODBC interface. Issues relating to security/password may have to be carefully dealt with because there will be many different customers hosting forms on the system.

Cumulative analysis

User submissions provide a tremendous wealth of data which companies rarely end up exploiting effectively. The forms are just accumulated with no attempt at looking for big picture patterns. This is usually because the additional step of importing the form submissions into a database or other system capable of performing analysis is too cumbersome. Also, generating reports on a regular basis is difficult to set up.
The present invention allows the customer to perform log analysis on demand and on a regular basis.

The following aspects of this functionality can be provided:

- A set of pre-canned queries, applicable across all forms is provided which may include:
  - Number of submissions per unit time. This can be a simple count of submissions per day/month, etc. For example, this may be useful as gross measure of the leads generated in a sales contact form.
  - Counts of radio's, checkboxes, selects. Items that have a fixed range of values can be counted. For example, if a select box asks "which financial management product are you interested in", then the number of submissions of each option (e.g., "product A" or "product B") can be tabulated.
  - Combination queries can be provided between various fixed input range fields. For example, if the user's income level is chosen in a select box and the automobile of interest is also a select (or radio), then a query can be set up that asks "What percent of users earning >$100,000 expressed an interest in automobile A"?
  - User IP address/hostname: the IP address of the user submitting the form can be resolved into hostnames. The number of submissions from various hosts or top level domains can be tabulated.
- The user can set up arbitrary queries between any form fields. In one embodiment, this feature can be provided via a user interface that allows the user to compose the query and also shows the resulting database query (e.g. SQL) statement for sophisticated users that would like to edit the statement.
- The analysis can be performed on demand, interactively at any time
- The set of queries can be stored, recalled, edited, deleted, etc.
- Any query or group of queries can be set up to be run at a regular interval, e.g., every night or every month. The query results can be sent to an email address.

Per submission management

In one embodiment, a complete interface for a customer to track and manage each user submission can be provided. Some of the features of this functionality are listed below:

- The customer can list his full set of forms, then ask for a list of submissions of a particular form. In one embodiment, mechanisms to scroll through the list through

multiple pages if the number of submissions is large can be provided (e.g., similar to message boards).

- The customer can ask to see detail on a particular form. In one embodiment, this leads to a page where the full table row associated with the form is displayed.
- At configuration time the customer may set up certain extra fields associated with each form. These fields amount to extra columns in the database table. In one embodiment, these fields do not receive values from user with the submitted form. The idea is for the customer to enter these values at a later time. The usefulness of this feature is illustrated in the examples below.
- The customer can change the value of the extra fields. The extra fields might be textual fields, checkboxes, radios, selects, etc., essentially all the usual input types of a form.
- The customer can also change the values of the fields that came in with the form. This allows editing of certain errors, etc.
- The customer can delete a particular submission entirely.
- This will involve writes to the database, but on a much lower performance path than the writes done at submit time.

As an example, consider a tech support form which has the usual customer information and problem definition fields. The customer might choose to define 3 additional "extra fields". These could be 'notes' for textual entry of notes by the person handling the case: 'case status' - a select with 3 possibilities open, closed, escalated; and 'person handling the case' - again a select with 2 possibilities Joe and Mary. Thus, Joe can come in to the list of submissions, select a particular one, make a change to the 'person handling the case' to indicate that he is the one who handles the case, call the customer and make some notes in the 'notes' field, and ultimately change the case status to 'closed'. Joe's supervisor can then come in and make a query on how many cases Joe closed last week.

Email

Email is a powerful additional service provided by the system according to the teachings of the present invention to enable a component of a business process.

The email capability has the following aspects:

- When the email is sent
- What the target addresses are
- What the email content is

In one embodiment, the customization of the functionality can be done through a browser interface. These various aspects of the email capability are described in more detail below.

When email is sent

- A customer may choose to send email on demand by manual action. The actual delivery of the email may be specified to be
  - Now
  - Delayed to a later time
- The customer may set up an automatic email delivery based on events. When a first submission is received, a subsequent event or action can be set up for this submission.

  These events may include the following:
  - An email can be generated to the user who submitted the form. For example, this could be a "reminder" email – telling a new subscriber to a service to check out the wonderful new stuff available at a web site
  - An email can be generated to the customer. For example, this can serve as a reminder to a sales person to ensure that a follow up happens to a sales lead within 5 days of form submission
  - A new web page form can be generated as a "follow up" form. For example, customers who are prospects can be emailed a URL to come back to. This form would make use of cookies to recognize a returning user, and correlate the user submission of this new form with the previous submission.
  - The events/actions can be scheduled:
    - At a particular time of the month
    - A certain number of days after the first submission
    - Switched on/off by other automatic or manual events that occur in the mean time

Target Addresses

- The target address can be the complete set of email addresses captured in a table of user submissions.
- The target address can be a subset of the email addresses in a table. The subsets can be:
  - Only those submissions within a certain date range (newer than x, older than x, between x and y, etc.)
  - Only those submissions with some other property set in the table, for example, only those who requested widget blue and not to those who requested widget red.
- The target could be a single address taken from the current form.

Email content

- The content may be a fixed, pre-determined message.
- The message may be uploaded by the customer.
- The message may be entered in a text box by the customer.
- The message may be "mail merged" to be customized to each addressee.

Figure 6 shows a flow diagram of one embodiment of a method according to the teachings of the present invention. The method starts at block 601 and proceeds to block 610. At block 610, a form is obtained from an author. At block 620, a parsing function is performed to extract attribute information with respect to various form elements contained in the form. At block 630, the author is allowed to configure a set of actions that are to be performed in processing submissions of the form. As described above, the set of actions to be performed in processing submissions of the form include form field validation or validation of user input, form response, form emailing, form logging, form threading, derived quantity generation, license generation, cookies generation and tracking, etc. At block 640, a configuration data structure or control file is generated to store the configuration information provided by the author. As described above, the configuration information provided by the author is used to configure the functionality of a specific program that is designated as the program to process the submissions of the form. In another embodiment, the configuration information provided by the author is used to configure a set of directives to be used by a general form processing program in processing submissions of a plurality of forms.

The invention has been described in conjunction with the preferred embodiment. It is evident that numerous alternatives, modifications, variations and uses will be apparent to those skilled in the art in light of the foregoing description. Although the invention has been described with reference to specific exemplary embodiments, it will be evident that various modifications and changes may be made to these embodiments without departing from the broader spirit and scope of the invention. Accordingly, the specification and drawings are to be regarded in an illustrative sense rather than a restrictive sense.

CLAIMS

What is claimed is:

1. A method comprising:

receiving a form from a user; and

allowing said user to configure a set of actions to be performed in processing a submission of said form, comprising:

parsing said form to extract specification of form elements contained in said form;

presenting said user with a user interface to allow said user to input specification for said set of actions to be performed in processing said submission of said form;

obtaining specification for said set of actions from said user; and

generating a configuration structure corresponding to said specification for said set of actions to be performed.

2. The method of claim 1 further comprising:

generating a program whose functionality corresponds to said set of actions specified by said user, said program to process submissions of said form.

3. The method of claim 1 wherein said user interface is generated based upon specification of said form obtained from parsing said form.

4. The method of claim 3 wherein said user interface comprises one or more Java applets.

5. The method of claim 3 wherein said user interface comprises one or more HTML pages.

6. The method of claim 1 further comprising:

maintaining consistency between said configuration structure and said form.

7. The method of claim 6 wherein maintaining comprises:

allowing said user to keep track of changes to said form after said set of actions was configured; and

allowing said user to reconfigure said set of actions based upon corresponding changes that have been made to said form.

8. The method of claim 6 wherein maintaining comprises:

determining whether said form has changed from the time said set of actions was configured, comprising:

storing an original timestamp associated with said form when said set of actions was configured; and

periodically checking a current timestamp associated with said form and comparing the current timestamp with the original time stamp to determine whether said form has changed from the time said set of actions was configured; and

if said form has changed, indicating that said form and said configuration structure are out of sync.

9. The method of claim 8 further comprising:

notifying said user that said form and said configuration structure are out of sync; and

allowing said user to reconfigure said set of actions for the changed form.

10. The method of claim 6 wherein maintaining comprises:

determining whether said form has been changed with respect to said form elements contained in said form;

indicating that said form and said configuration are out of sync if said form has been changed with respect to said form elements.

11. The method of claim 10 wherein determining comprises:

determining whether a new form element has been added to said form.

12. The method of claim 10 wherein determining comprises:

determining whether an existing form element has been removed from said form.

13. The method of claim 10 wherein determining comprises:

determining whether one or more attributes with respect to an existing form element has been changed.

14. The method of claim 10 further comprising:

determining whether changes to said form result in an incompatible change in the corresponding form processing program; and

if the changes to said form result in an incompatible change in the corresponding form processing program, notifying said user that said form and the corresponding processing program are no longer compatible and allowing said user to redo the configuration for the changed form.

15. The method of claim 14 further comprising:

if the changes to said form do not result in an incompatible change in the corresponding form processing program, continuing to perform those actions that are not affected by the changes in processing submissions of the changed form.

16. The method of claim 6 wherein maintaining comprises:

receiving a submission of said form from a user;

extracting current instance of said form elements and their submitted values from said submission of said form;

comparing said current instance of said form elements with previous instance of said form elements stored in said configuration structure; and

if said current instance of said form elements is different with respect to said previous instance of said form elements, indicating that said form and said configuration structure are no longer consistent.

17. The method of claim 16 further comprising:

determining whether changes to said form result in an incompatible change in the corresponding form processing program; and

if the changes to said form result in an incompatible change in the corresponding form processing program, notifying said user that said form and the corresponding processing program are no longer compatible and allowing said user to redo the configuration for the changed form.

18. The method of claim 17 further comprising:

if the changes to said form do not result in an incompatible change in the corresponding form processing program, continuing to perform those actions that are not affected by the changes in processing submissions of the changed form.

19. A method comprising:

receiving a first form created by a first author, the first form including one or more input fields, each input field having one or more field attributes;

parsing the first form to extract specification information with respect to the one or more input fields including the one or more field attributes associated with each input field; and

presenting to the first author a user interface including a set of questions that are constructed based upon the extracted specification information with respect to the one or more input fields, the user interface allowing the first author to specify one or more specific actions that are to be performed in processing submissions of the first form.

20. The method of claim 19 further comprising:

performing the one or more specific actions specified by the first author in processing submissions of the first form.

21. The method of claim 19 further comprising:

generating a first program including one or more program components to perform the one or more actions specified by the first author in processing a submission of the first form submitted by a first user, the first program being designated as the program to process submissions of the first form.

22. The method of claim 19 further comprising:

generating a set of directives corresponding to the one or more actions specified by the first author, the set of directives is to be used by a first program that is designated as the program to process a plurality of forms including the first form.

23. The method of claim 22 wherein the first program is configured to process multiple forms based upon multiple sets of directives, each set of directives corresponding to each form.

24. The method of claim 19 wherein a configuration data structure is created for the first form to include configuration information provided by the first author with respect to the set of functions to be performed in response to a submission of the first form.

25. The method of claim 19 further comprising:

maintaining consistency between the configuration data structure and the specification of the first form.

26. The method of claim 25 wherein maintaining comprises:

modifying the configuration information in the configuration data structure in response to changes in the specification of the first form.

27. The method of claim 25 wherein maintaining consistency comprises:

determining whether the first form has been changed since the configuration data structure was created; and

if the first form has been changed since the configuration data structure was created, updating the configuration data structure to reflect the changes that have been made to the first form.

28. A program comprising:

first programming logic to parse a first form provided by a first author to extract information with respect to a set of input fields included in the first form, the information including attributes of the input fields; and

second programming logic to present to the first author a first user interface including a list of questions generated, the first user interface allowing the first author to provide configuration information with respect to a set of actions to be performed in response to a submission of the first form.

29. The program of claim 28 wherein the list of questions is generated based upon the extracted information.

30. The program of claim 28 wherein the configuration information for the set of actions to be performed is stored in a configuration data structure.

31. The program of claim 28 further comprising:

third programming logic to generate a set of actions based upon the configuration information provided by the first author, the actions are to be performed in connection with the processing of a submission of the first form from a user.

32. The program of claim 31 wherein the set of actions is to be included in a program that is designated to process submissions of the first form when the submissions of the first form are submitted for processing.

33. The program of claim 31 wherein the set of actions are designated as a set of directives to be used by a program for processing submissions of the first form, the program being designated to process submissions of multiple forms based upon multiple sets of directives each corresponding to each form.

34. The program of claim 31 further comprising:

fourth programming logic to maintain consistency between the configuration information provided by the first author and the first form.

35. The program of claim 34 wherein fourth programming logic comprises:

logic to detect changes to the first form after the configuration information is provided by the first author; and

logic to update the configuration information to reflect changes made to the first form.

36. A system comprising:

a first module to parse a first form created by a first author to extract specification information with respect to a set of form elements contained in the first form; and

a second module to obtain from the first author configuration information for a set of actions to be performed in processing a submission of the first form and to configure a first program using the configuration information obtained from the first author, the first program being designated as a processing program for the first form.

37. The system of claim 36 wherein the second module presents to the first author a list of questions constructed based upon the specification information extracted from the first form.

38. The system of claim 37 wherein the questions are used to obtain the configuration information from the first author with respect to the set of actions to be performed in processing a submission of the first form.

39. The system of claim 36 wherein, after obtaining the configuration information from the first author, the second module generates a set of functions to be included in the first program in processing submissions of the first form.

40. The system of claim 36 wherein, after obtaining the configuration information from the first author, the second module generates a set of processing directives to be used by a second program for processing submissions of the first form, the second program being designated as a processing program for multiple forms including the first form.

41. The system of claim 36 further comprising:

logic to maintain consistency between the specification of the first form and the functionality of the first program.

42. The system of claim 41 wherein logic to maintain consistency comprises:

logic to detect subsequent changes made to the first form; and

logic to update the first program's functionality to reflect the subsequent changes made to the first form.

43. A method comprising:

allowing a first author of a first form who is located at a first location to remotely configure a first program's functions at a second location, the first program to perform its corresponding functions in processing submissions of the first form, said allowing comprising:

receiving the first form at a first server;

parsing the first form to extract specification information of form elements included in the first form;

creating a representation of the form elements based upon specification information extracted from the first form;

obtaining configuration information from the first author; and

configuring the first program's functions using the configuration information provided by the first author.

44. The method of claim 43 wherein the representation of the form elements is used to obtain configuration information from the first author.

45. The method of claim 43 wherein the functions to be performed by the first program comprise a field validation function that includes validating one or more form elements included in the first form.

46. The method of claim 43 wherein configuring the first program's functions comprises:

obtaining validation criteria for the one or more form elements from the first author;

configuring the field validation function based upon the validation criteria obtained from the first author.

47. The method of claim 43 wherein the functions to be performed by the first program include a derived quantity generation function that includes generating one or more quantities based upon a user's submitted values with respect to one or more form elements contained in the first form.

48. The method of claim 47 wherein configuring the first program's functions comprises:

obtaining the configuration information from the first author with respect to the derived quantity generation function; and

configuring the derived quantity generation function based upon the configuration information obtained from the first author.

49. The method of claim 43 wherein the functions to be performed by the first program include a license generation function that includes generating one or more licenses in response to the first form's submission.

50. The method of claim 49 wherein configuring the first program's functions comprises:

obtaining the license configuration information from the first author; and

configuring the license generation function based upon the license configuration information obtained from the first author.

51. The method of claim 43 wherein the functions to be performed by the first program include a cookie generation function that includes generating one or more cookies for each user who submits a submission of the first form for processing.

52. The method of claim 51 wherein configuring the first program's functions comprises:

obtaining the cookie configuration information from the first author with respect to the cookie generation function; and

configuring the cookie generation function based upon the cookie configuration information obtained from the first author.

53. The method of claim 43 wherein the functions to be performed by the first program include a form emailing function that includes emailing the contents of the first form's submission to one or more addresses specified by the first author in response to the submission of the first form by a user.

54. The method of claim 53 wherein configuring the first program's functions comprises:

obtaining the email configuration information from the first author with respect to the form emailing function; and

configuring the form emailing based upon the email configuration information obtained from the first author.

55. The method of claim 43 wherein the functions to be performed by the first program include an email-to-the-user function that includes sending an email message to a user who submits the first form.

56. The method of claim 55 wherein configuring the first program's functions comprises:

obtaining the configuration information from the first author with respect to the email-to-the-user function; and

configuring the email-to-the-user function based upon the configuration information obtained from the first author.

57. The method of claim 43 wherein the functions to be performed by the first program include a form response function that includes generating one or more web pages in a user's browser. upon receipt of the first form's submission from the user.

58. The method of claim 57 wherein configuring the first program's functions comprises:

obtaining the form response configuration information from the first author with respect to the form response function; and

configuring the form response generation function based upon the configuration information obtained from the first author.

59. The method of claim 43 wherein the functions to be performed by the first program include a form threading function that includes preserving a state of one or more fields in the first form and passing the state of the one or more fields in the first form to a second form.

60. The method of claim 59 wherein configuring the first program's functions comprises:

obtaining the configuration information from the first author with respect to the form threading function; and

configuring the form threading function based upon the configuration information obtained from the first author.

61. The method of claim 60 wherein the functions to be performed by the first program include a form logging function that includes logging a user's submitted values for one or more form elements included in the first form.

62. The method of claim 61 wherein the submitted values are stored in a database.

63. The method of claim 62 wherein the values in each submission are stored in a single row of a table, different submissions corresponding to different rows of the table.

64. The method of claim 61 wherein configuring the first program's functions comprises:

obtaining the configuration information from the first author with respect to the form logging function; and

configuring the form logging function based upon the configuration information obtained from the first author.

65. The method of claim 64 further comprising:

providing the first author with a capability to manage data relating to the submissions of the first form by one or more users.

66. The method of claim 65 wherein providing comprises:

logging data with respect to the first form's submissions by the one or more users; and

allowing the first author to access and use the logged data.

67. The method of claim 65 further comprising:

providing the first author with a capability to perform a cumulative analysis function with respect to the data received from submissions of the first form.

68. The method of claim 67 wherein providing comprises:

allowing the first author to construct a set of queries to extract one or more types of information with respect to the submissions of the first form.

69. The method of claim 68 wherein the first author is allowed to perform the cumulative analysis function on a regular basis.

70. The method of claim 68 wherein the first author is allowed to perform the cumulative analysis function on demand.

71. The method of claim 65 wherein the first author is allowed to track and manage each user's submission of the first form.

72. The method of claim 69 wherein the first author is allowed to select a particular submission of the first form from a list of submissions of the first form and to modify the submitted value of one or more form elements in the first form.

73. The method of claim 72 wherein the first author is allowed to add extra data fields associated with each submission, the values for these fields being provided separately by the first author and separate from the submission.

74. The method of claim 43 further comprising:

allowing the first author to configure an email function to be performed by the first program in response to a submission of the first form by a user.

75. The method of claim 74 wherein allowing the first author comprises:

allowing the first author to specify when an email message is to be sent;

allowing the first author to specify to one or more target addresses to which the email message is to be sent; and

allowing the first author to specify the content of the email message.

76. A machine-readable medium comprising instructions which, when executed by a machine, cause the machine to perform operations comprising:

receiving a form from a user;

parsing said form to extract specification of form elements contained in said form; and

allowing said user to configure a set of actions to be performed in processing a submission of said form, comprising:

presenting said user with a user interface to allow said user to input specification of said set of actions to be performed in processing said submission of said form;

obtaining specification for said set of actions from said user; and

generating a configuration structure corresponding to said specification for said set of actions to be performed.

77. The machine-readable medium of claim 76 further comprising:

generating a program whose functionality corresponds to said specification of said set of actions, said program to process submissions of said form.

78. The machine-readable medium of claim 76 further comprising:

generating a set of directives corresponding to said specification of said set of actions, said set of directives is to be used by program that is designated as the program to process a plurality of forms.

79. The machine-readable medium of claim 76 wherein said user interface is generated based upon specification of said form obtained from parsing said form.

80. The machine-readable medium of claim 76 further comprising:

maintaining consistency between said configuration structure and said form.

81. A machine-readable medium comprising instructions which, when executed by a machine, cause the machine to perform operations comprising:

receiving a first form created by a first author, the first form including one or more input fields, each input field having one or more field attributes;

parsing the first form to extract specification information with respect to the one or more input fields including the one or more field attributes associated with each input field; and

presenting to the first author a user interface including a set of questions with respect to the one or more input fields, the user interface allowing the first author to specify one or more specific actions that are to be performed in processing submissions of the first form.

82. The machine-readable medium of claim 81 wherein the set of questions is constructed based upon the extracted specification information.

83. The machine-readable medium of claim 81 further comprising:

performing the one or more specific actions specified by the first author in processing submissions of the first form.

84. The machine-readable medium of claim 81 further comprising:

generating a first program including one or more program components to perform the one or more actions specified by the first author in processing a submission of the first form submitted by a first user, the first program being designated as the program to process submissions of the first form.

85. The machine-readable medium of claim 81 further comprising:

generating a set of directives corresponding to the one or more actions specified by the first author, the set of directives is to be used by a first program that is designated as the program to process a plurality of forms including the first form.

86. The machine-readable medium of claim 81 wherein a configuration data structure is created for the first form to include configuration information provided by the first author with respect to the set of actions to be performed in processing submissions of the first form.

87. The machine-readable medium of claim 86 further comprising:

maintaining consistency between the configuration data structure and the first specification of the first form.

88. The machine-readable medium of claim 87 wherein maintaining comprises:

modifying the configuration information in the configuration data structure in response to changes in the specification of the first form.

89. A machine-readable medium comprising instructions which, when executed by a machine, cause the machine to perform operations comprising:

allowing a first author of a first form who is located at a first location to remotely configure a first program's functions at a second location, the first program to perform its

corresponding functions in processing submissions of the first form, said allowing comprising:

receiving the first form at a first server;

parsing the first form to extract specification information of form elements included in the first form;

creating a representation of the form elements based upon specification information extracted from the first form;

obtaining configuration information from the first author; and

configuring the first program's functions using the configuration information provided by the first author.

90. The machine-readable medium of claim 89 wherein the functions to be performed by the first program comprise a field validation function that includes validating one or more form elements included in the first form.

91. The machine-readable medium of claim 89 wherein the functions to be performed by the first program include a derived quantity generation function that includes generating one or more quantities based upon a user's submitted values with respect to one or more form elements contained in the first form.

92. The machine-readable medium of claim 89 wherein the functions to be performed by the first program include a license generation function that includes generating one or more licenses in response to the first form's submission.

93. The machine-readable medium of claim 89 wherein the functions to be performed by the first program include a cookie generation function that includes generating one or more cookies for each user who submits the first form for processing.

94. The machine-readable medium of claim 89 wherein the functions to be performed by the first program include a form emailing function that includes emailing the contents of the first form's submission to one or more addresses specified by the first author in response to the submission of the first form by a user.

95. The machine-readable medium of claim 89 wherein the functions to be performed by the first program include an email-to-the-user function that includes sending an email message to a user who submits the first form.

96. The machine-readable medium of claim 89 wherein the functions to be performed by the first program include a form response function that includes generating one or more responses upon receipt of the first form's submission from a user.

97. The machine-readable medium of claim 89 wherein the functions to be performed by the first program include a form threading function that includes preserving a state of one or more fields in the first form and passing the state of the one or more fields in the first form to a second form.

98. The machine-readable medium of claim 89 wherein the functions to be performed by the first program include a form logging function that includes logging a user's submitted values for one or more form elements included in the first form.

99. The method of claim 98 wherein the submitted values are stored in a database.

100. The method of claim 99 wherein the values in each submission are stored in a single row of a table, different submissions corresponding to different rows of the table.

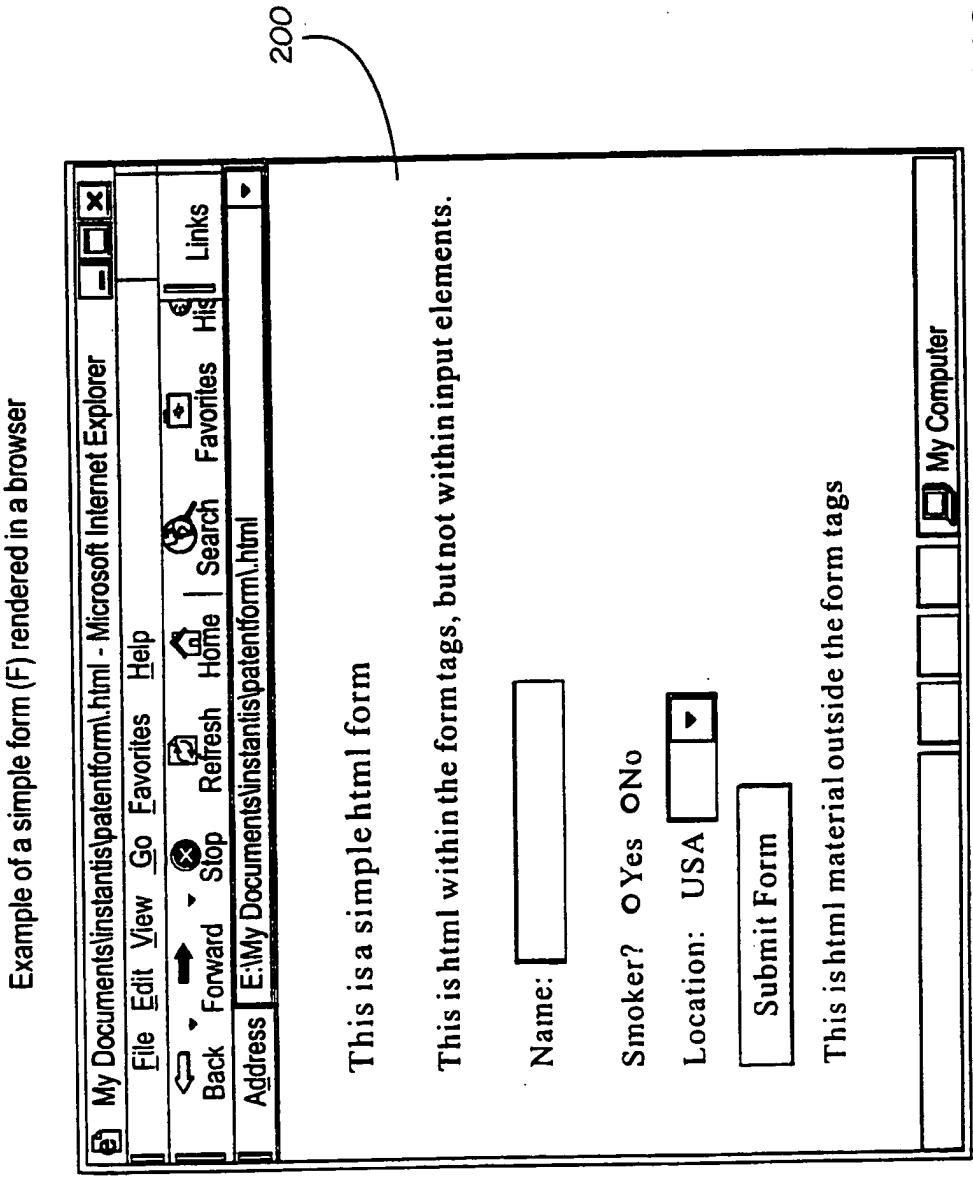
101. The machine-readable medium of claim 89 further comprising:

providing the first author with a capability to manage data relating to the submissions of the first form by one or more users.

Example of a simple HTML form (F) showing html code

```
<head>
</head>
<body>
<h3>This is a simple html form</h3>
<form method=POST action="http://www.remoteserver.com/form.cgi">

This is html within the form tags, but not within input elements.<p>

Name: <input type=text name=firstname size=10><p>

Smoker?: <input type=radio name=smoker value=yes> Yes  
         <input type=radio name=smoker value=no No<p>

Location:
<select name=location>
<option value=US>USA
<option value=ROW>Other
</select> <p>

<input type=hidden name=id value=1002>
<input type=submit name=submit value="Submit Form">

</form>
This is html material outside the form tags.
</body>
```

FIG. 1



Example of a simple form (F) rendered in a browser

FIG. 2

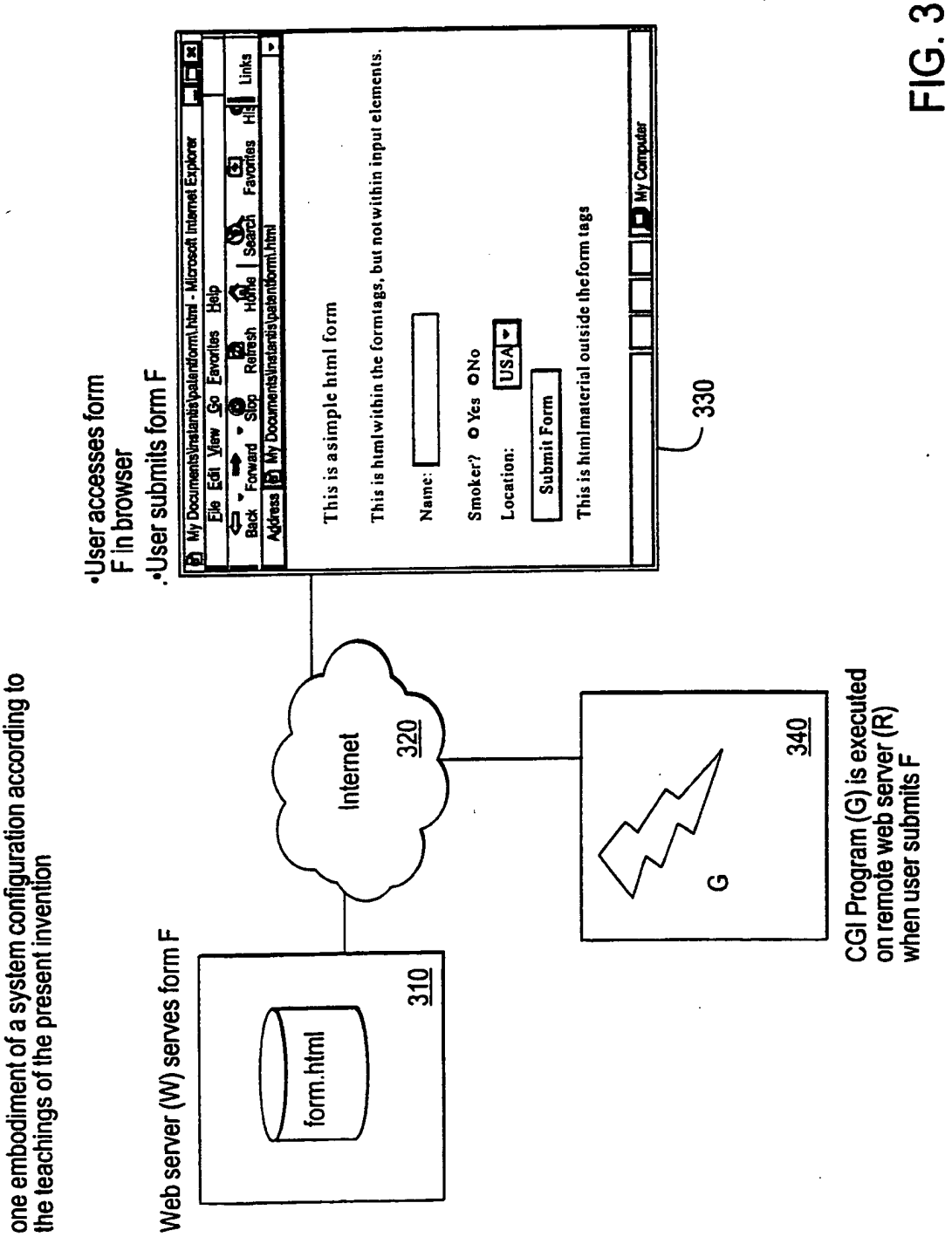

one embodiment of a system configuration according to the teachings of the present invention
Web server (W) serves form F
form.html
310
Internet
320
G
340
CGI Program (G) is executed on remote web server (R) when user submits F
•User accesses form F in browser
•User submits form F
330
This is a simple html form
This is html within the form tags, but not within input elements.
Name:
Smoker? Yes No
Location: USA
Submit Form
This is html material outside the form tags
My Computer

FIG. 3

Example of one embodiment of user interface for configuring form functionality

400

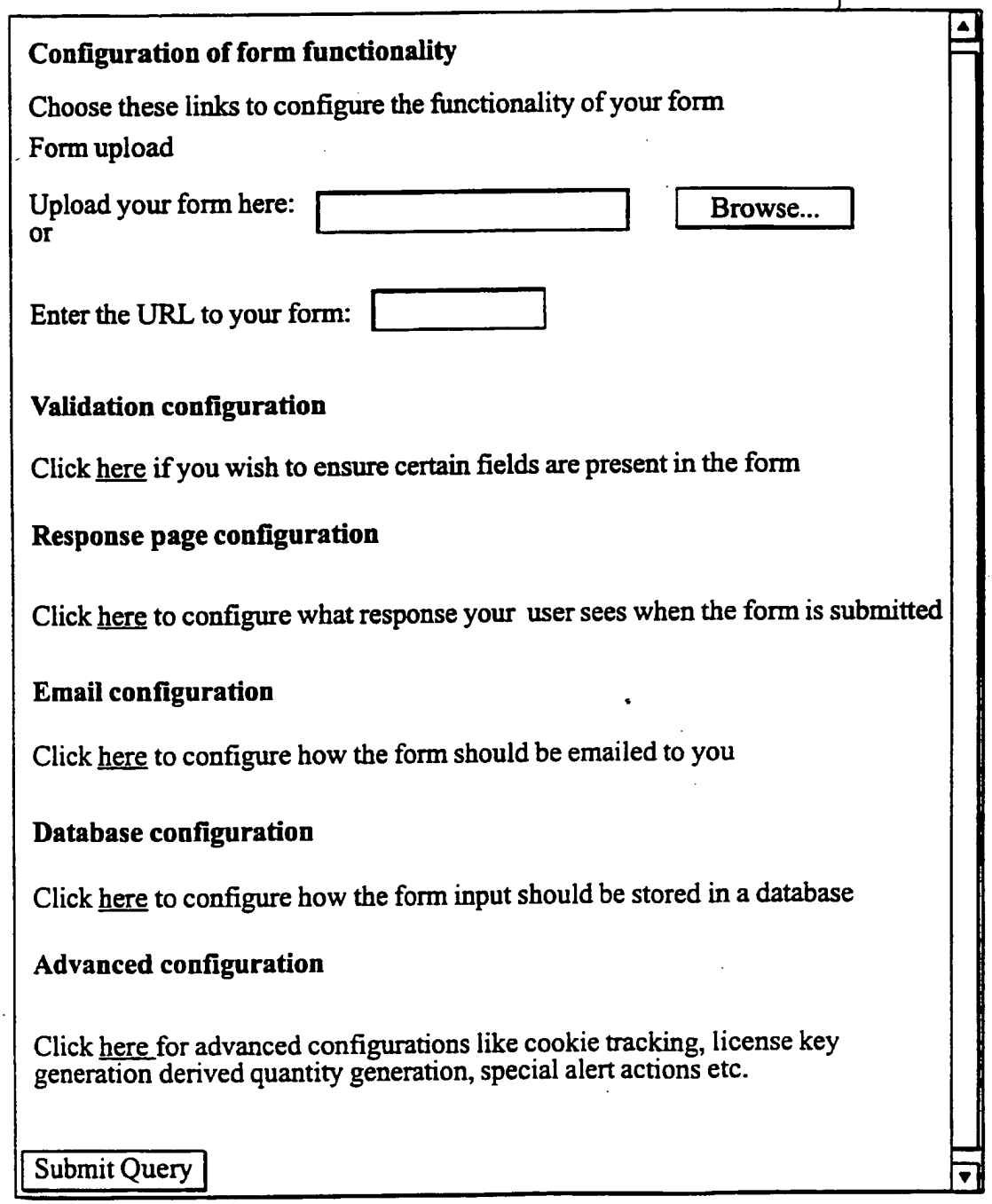


FIG. 4

5/6

block diagram of one embodiment of a system for remote configuration of forms and generation of cgi

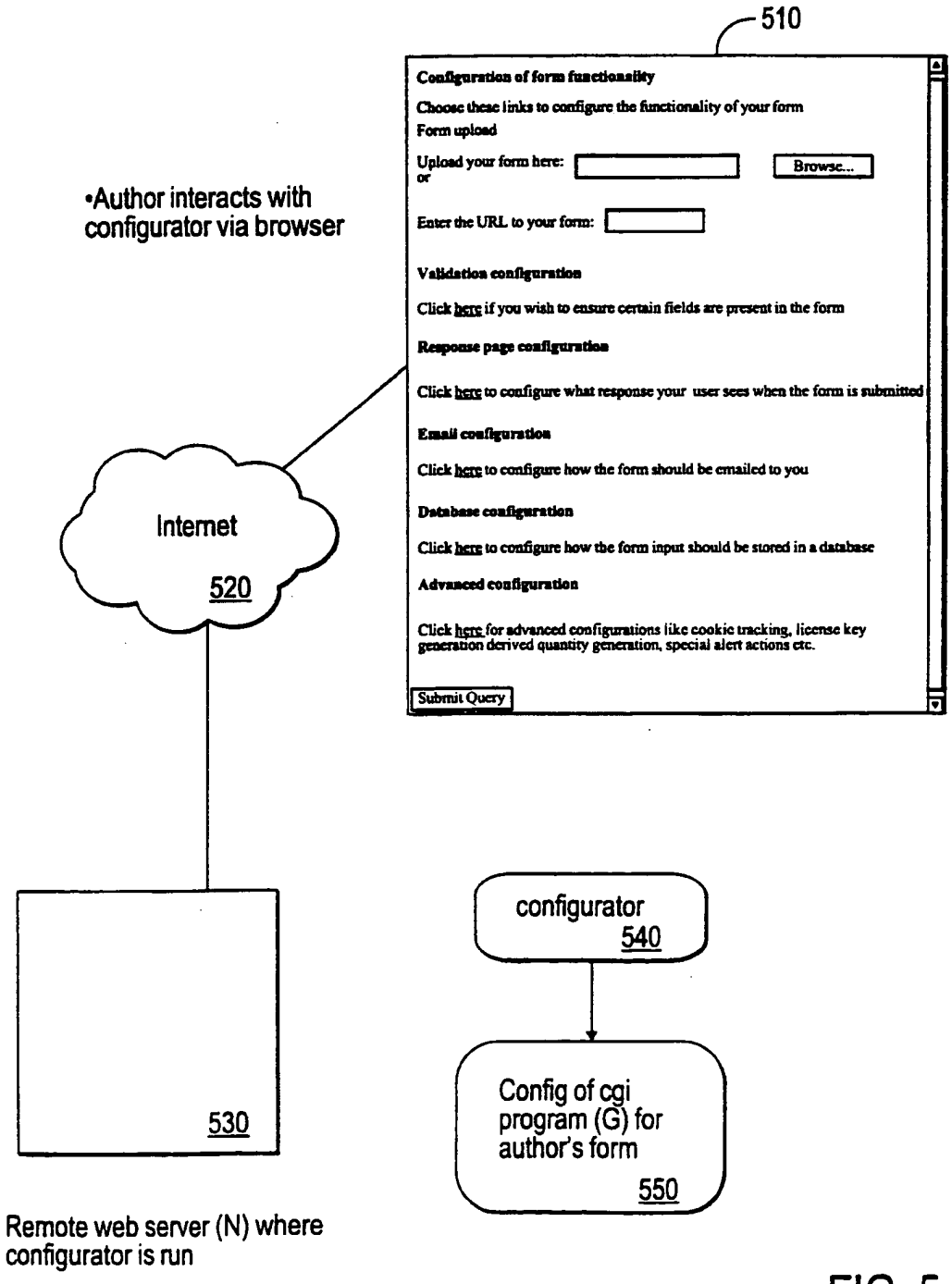


FIG. 5

flow diagram of one embodiment of a method for form configuration

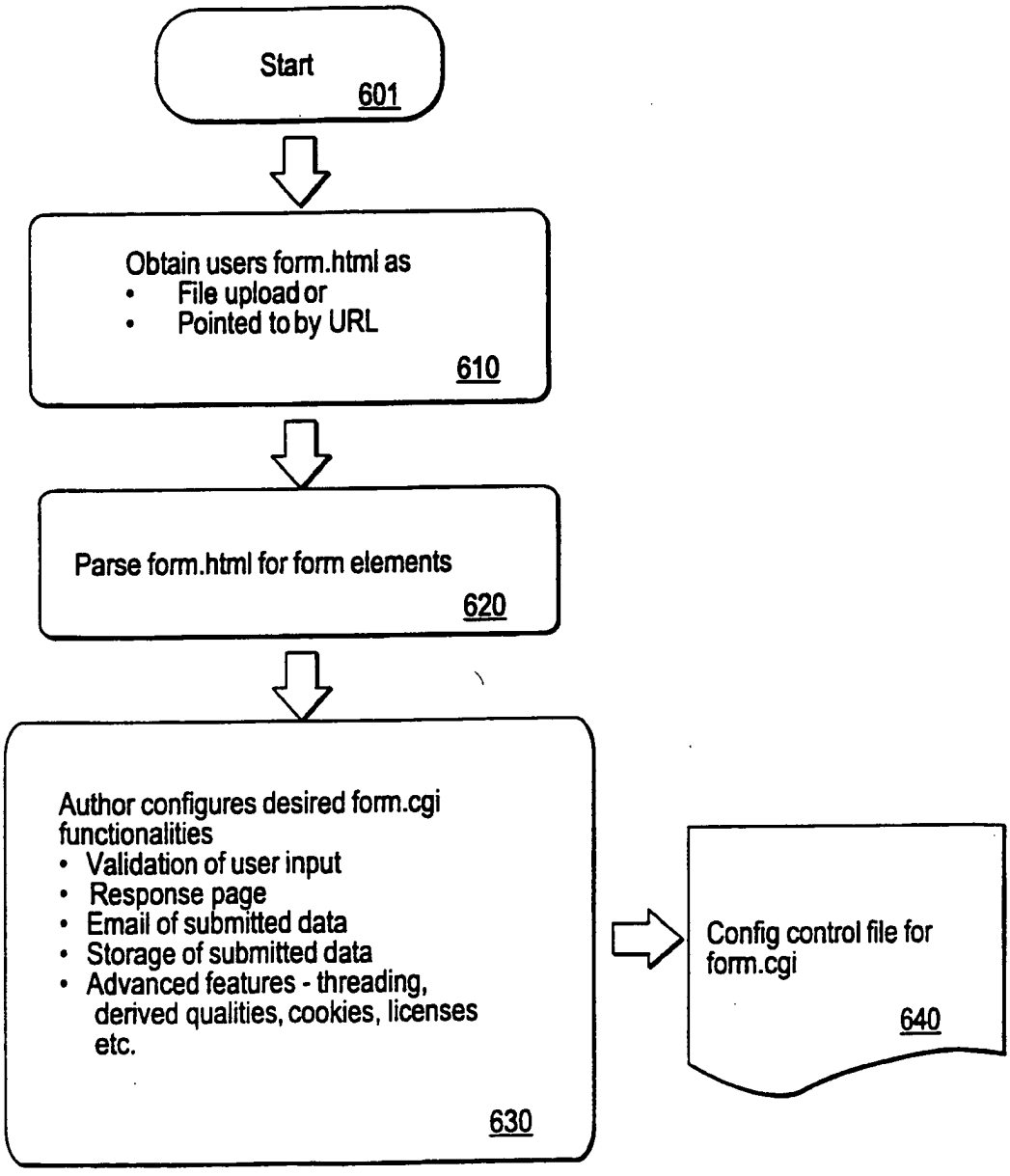


FIG. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/26883

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :GO6 F 17/60
US CL :705/1

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 705/1

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

STN

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y, P	US 6,005,597 A (BARRETT et al) 21 December 1999, abstract, figure 17, column 10, lines 60-67, column 11, lines 3-11	1-101

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 29 NOVEMBER 2000	Date of mailing of the international search report 30 JAN 2001
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer JAMES TRAMMEL Telephone No. (703) 305-9768